1998

Efficient Parallel Algorithms for Planar st-Graphs

Mikhail J. Atallah
Purdue University, mja@cs.purdue.edu

Danny Z. Chen

Ovidiu Daescu

Report Number:
98-021

EFFICIENT PARALLEL ALGORITHMS FOR PLANAR ST-GRAPHS

Mikhail J. Atallah
Danny Z. Chen
Ovidiu Daescu

CSD-TR #98-021
June 1998

Efficient Parallel Algorithms for Planar *st*-Graphs

Mikhail J. Atallah*
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907, USA
mja@cs.purdue.edu

Danny Z. Chen† and Ovidiu Daescu†
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
{chen,odaescu}@cse.nd.edu

Abstract

Planar *st*-graphs find applications in a number of areas. In this paper, we present efficient parallel algorithms for solving several fundamental problems on planar *st*-graphs. The problems we consider include all-pairs shortest paths in weighted planar *st*-graphs, single-source shortest paths in weighted planar layered digraphs (which can be reduced to single-source shortest paths in certain special planar *st*-graphs), and depth-first search in planar *st*-graphs. Our parallel shortest path techniques exploit the specific geometric and graphic structures of planar *st*-graphs, and involve schemes for partitioning planar *st*-graphs into subgraphs in a way that ensures that the resulting path length matrices have a monotonicity property [1, 2]. The parallel algorithms we obtain are considerable improvement over the previously best known solutions (when they are applied to these *st*-graph problems), and are in fact relatively simple. The parallel computational models we use are the CREW PRAM and EREW PRAM.

1 Introduction

An n-vertex planar *st*-graph $G = (V, E)$ is a planar directed acyclic graph with exactly one *source* vertex s and exactly one *sink* vertex t, such that G can be embedded in the plane with both s and t on the boundary of the external face of the embedding of G. Planar *st*-graphs find applications in a number of areas, including partial orders, computational geometry (e.g., path planning, planar point location, and visibility), graph theory (e.g., graph drawing, planar graph embedding, and graph planarization), VLSI design (e.g., floorplanning and layout compaction), and motion planning (see [32] for references). Series-parallel graphs are a special case of planar *st*-graphs.

In this paper, we present efficient deterministic parallel algorithms for solving several fundamental problems on planar *st*-graphs. In particular, we consider the following problems.

*This author gratefully acknowledges the support of the COAST Project at Purdue University and its sponsors, in particular Hewlett Packard, DARPA, and the National Security Agency.

†The research of these authors was supported in part by the National Science Foundation under the NSF Grant CCR-9623585.

- All-pairs shortest paths in weighted planar st-graphs (whose edges are associated with nonnegative weights).
- Single-source shortest paths in weighted planar layered digraphs. Note that shortest paths in planar layered digraphs can be reduced to shortest paths in certain special planar st-graphs.
- Depth-first search in planar st-graphs.

We henceforth assume all graphs have n vertices and have edges whose weights are nonnegative.

The parallel computational models we use are the CREW PRAM and EREW PRAM [14]. Recall that the PRAM is a synchronous parallel model in which all processors share a common memory and each processor can access any memory location in constant time. The CREW PRAM allows simultaneous accesses to the same memory location by multiple processors only if all such accesses are for reading data only. The EREW PRAM forbids multiple processors to simultaneously access the same memory location. We also refer to the CRCW PRAM model, which allows simultaneous accesses to the same memory location for both reading and writing data. Note that the CRCW PRAM is more powerful than the CREW PRAM and EREW PRAM, and the simulation of a CRCW PRAM algorithm on a CREW or EREW PRAM by using the same number of processors can cause an increase in the time bound by a logarithmic factor. For convenience, all our algorithms will be described using two parallel complexity bounds: Time and work (the *work* is the *time* $\times$ *processor* product of an algorithm). The processor bounds of all our parallel algorithms can be easily derived by using Brent's theorem [4].

The problem of computing shortest paths in graphs is fundamental in computer science, and has applications in solving many scientific and engineering problems. Sequentially, the problem is quite well studied and efficient algorithms for various versions of graph shortest path problems have been given (e.g., see [6, 8]). However, designing efficient parallel algorithms for computing shortest paths in graphs has remained an elusive task despite considerable efforts.

Several interesting parallel graph algorithms for all-pairs shortest paths are known. Han, Pan, and Reif [11] presented an $O(\log^2 n)$ time, $o(n^3)$ work EREW PRAM algorithm for all-pairs shortest paths in general directed graphs. Cohen [5] gave an $O(\log^4 n)$ time, $O(n^2 \log n)$ work CREW PRAM algorithm for all-pairs shortest paths in planar directed graphs. An $O(\log n)$ time, $O(n^2)$ work EREW PRAM algorithm for all-pairs shortest paths in series-parallel digraphs can be obtained from the results of [33] (by using the parallel tree contraction technique [14]). Note that a *series-parallel digraph* is a directed acyclic graph with exactly one source and exactly one sink, such that the graph can be constructed by series and parallel compositions. Series-parallel digraphs are a special case of planar st-graphs.

There are also parallel graph algorithms for single-source shortest paths. Pan and Reif [25, 26] developed an $O(\log^3 n)$ time, $O(n^{1.5})$ work algorithm for single-source shortest paths in planar undirected graphs, and this result has been generalized to planar directed graphs [5, 11]. Klein and Subramanian [21] gave a linear-processor, polylog-time algorithm for single-source shortest paths in planar directed graphs, but the exponent in their polylogarithmic running time is rather

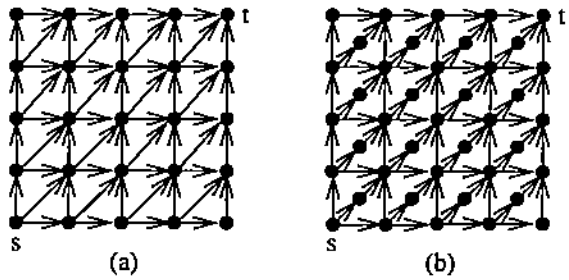

Figure 1: (a) A planar directed grid graph. (b) A corresponding planar layered digraph.

large. An $O(\log n)$ time, $O(n)$ work EREW PRAM algorithm for single-source shortest paths in series-parallel digraphs was presented in [33].

There are a few parallel algorithms for computing a shortest path between *one pair* of vertices in certain graphs. Aggarwal and Park [1] and Apostolico *et al.* [2] obtained an $O(\log^2 n)$ time, $O(n \log n)$ work CREW PRAM algorithm for finding a source-to-sink shortest path in planar directed acyclic grid graphs (see Figure 1(a) for an example of such graphs). An interesting generalization is the result of Sairam, Tamassia, and Vitter [28], who considered planar layered directed acyclic graphs (called planar layered digraphs). Note that the directed grid graphs considered in [1, 2] are in fact a special case of planar *st*-graphs, and can be transformed to planar layered digraphs (see Figure 1(b) for an example). The CREW PRAM algorithm in [28] finds a source-to-sink shortest path in planar layered digraphs in $O(\log^2 n)$ time and $O(n \log n)$ work, based on an elegant *one-way separator* construction. It was also shown in [28] that shortest paths in planar layered digraphs can be reduced to shortest paths in some special planar *st*-graphs.

The problem of computing a depth-first search tree in a graph is fundamental in graph theory, and is easily solved in linear time sequentially [6]. In parallel, performing ordered depth-first search that visits vertices of a graph in a given order is known to be P-complete [27]. There are parallel algorithms for depth-first search in various special graphs, such as chordal graphs [20], series-parallel graphs [12], planar undirected graphs [10, 13, 15, 19, 29, 30], and planar directed graphs [16, 17, 18]. In particular, Kao and Klein [18] gave an $O(\log^{10} n)$ time, $O(n \log^{10} n)$ work algorithm for depth-first search in general planar directed graphs, and Kao [17] presented an $O(\log^5 n)$ time, $O(n \log^4 n)$ work algorithm for depth-first search in strongly connected planar directed graphs.

In this paper, we present the following efficient parallel solutions:

- An $O(\log^2 n)$ time, $O(n^2)$ work CREW PRAM algorithm for all-pairs shortest paths in planar *st*-graphs. The techniques of this algorithm, when applied to computing all-pairs shortest paths in the planar directed acyclic grid graphs of [1, 2], yield an $O(\log^2 n)$ time, $O(n^2)$ work CREW PRAM algorithm.

- An $O(\log^2 n)$ time, $O(n \log n)$ work CREW PRAM algorithm for single-source shortest paths in planar layered digraphs. This generalizes the source-to-sink shortest path result of [28].

- An $O(\log n)$ time, $O(n)$ work EREW PRAM algorithm for depth-first search in planar *st*-graphs.

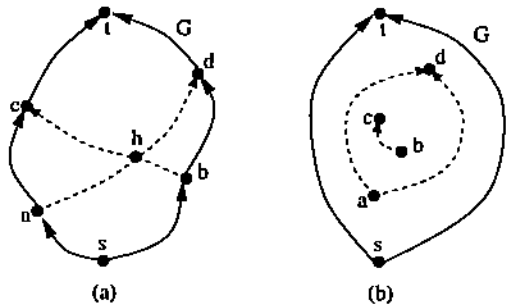

Figure 2: (a) Paths that must cross each other. (b) Paths that need not cross each other.

In comparison, our parallel algorithms are considerable improvement over the previously best known results (which are often for more general classes of graphs) when they are applied to these st-graph problems, and in fact are relatively simple. Further, note that our parallel depth-first search and all-pairs shortest path algorithms are optimal, and the work bound of our parallel single-source shortest path algorithm is within only a $\log n$ factor of the time bound of its corresponding optimal sequential algorithm.

All our parallel algorithms exploit the specific geometric and graphic structures of planar st-graphs. Our parallel shortest path techniques involve two graph partitioning schemes: The *strict one-way separators* for planar st-graphs (which make use of the visibility representation of planar st-graphs in [32]), and the *one-way separators* for planar layered digraphs (which are a slight modification of the ones used in [28]). These schemes partition planar st-graphs or planar layered digraphs into subgraphs in a way that ensures that the resulting path length matrices have a monotonicity property. This monotonicity property enables one to perform matrix multiplications in a very efficient manner. In particular, we get around the following difficulty that arises in all our shortest path computation. Observe that for the four vertices a, b, c, and d that are all on the boundary of the external face of an embedded planar st-graph G as shown in Figure 2(a), the shortest a-to-d path in G must cross the shortest b-to-c path in G at a vertex h (because G is a planar st-graph). Such a "crossing" property is a key to obtaining matrices with monotonicity property. However, when some of the vertices are not on the boundary of the external face of G (as shown in Figure 2(b)), shortest paths in G between these vertices need not cross each other. In all our shortest path computation (especially for the all-pairs case), we must use appropriate graph partitioning schemes for achieving path length matrices that have the monotonicity property.

For our parallel shortest path algorithms, we will only describe the versions for computing the *lengths* of the shortest paths. The length versions of the algorithms can be easily modified to generate actual shortest path trees as well as the shortest path lengths.

The rest of the paper is organized as follows. Section 2 reviews some useful definitions and preliminary results. Section 3 gives our parallel algorithm for all-pairs shortest paths in planar st-graphs. Section 4 presents our parallel algorithm for single-source shortest paths in planar layered digraphs. Section 5 discusses our parallel depth-first search algorithm on planar st-graphs.

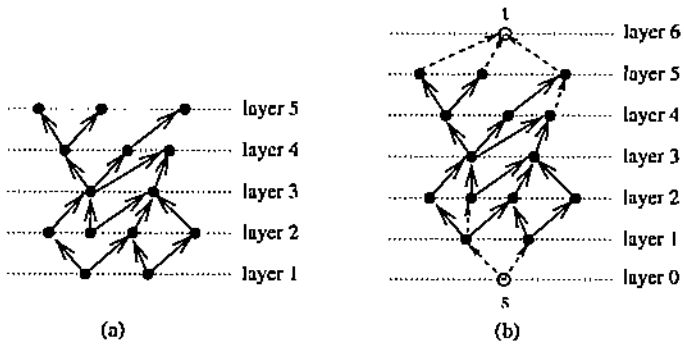

Figure 3: (a) A 5-layer planar layered digraph. (b) The corresponding planar layered st-graph.

2 Preliminaries

Let $G = (V, E)$ be an n-vertex input graph (either a planar st-graph or a planar layered digraph), with edges whose weights are nonnegative. As in [32], we assume that the input graph representation is embedded, that is, for each vertex v of G, the cyclic ordering of v's neighboring vertices (both incoming and outgoing) in the embedding is given in standard form (as a doubly-linked edge list). Without loss of generality (WLOG), we assume that the embedding of G is such that all its edges are directed upwards (e.g., by rotating the embedding of the graphs in Figure 1 by $\pi/4$).

Planar layered digraphs are also called as *planar proper hierarchies* [7, 34]. The definition of a planar layered digraph is reviewed as follows.

Definition 1 *A planar layered digraph is a planar directed acyclic graph $G = (V, E)$ that allows an embedding in the plane, called* k-line embedding, *that has the following properties:*

(i) The vertices of G are partitioned into k subsets called layers.

(ii) The k layers of G are consecutively embedded in k parallel lines, with layer i on the i-th line.

(iii) Every edge of G corresponds to a directed straight line segment from a vertex of layer i to a vertex of layer $i+1$, for some i with $1 \leq i < k$.

(iv) No two edges of G cross each other in the embedding.

A planar layered digraph may have multiple sources and sinks (e.g., see Figure 3(a)). Note that planar layered digraphs are a generalization of planar directed acyclic grid graphs in the sense that such a grid graph can be easily transformed to a planar layered digraph (e.g., see Figure 1).

It has been shown in [28] that given a k-line embedding of an n-vertex planar layered digraph G, it is possible to transform G to an $(n+2)$-vertex planar layered st-graph G' with a $(k+2)$-line embedding, such that shortest path problems in G are equivalent to shortest path problems in G' (this is done by adding to G a new source s and a new sink t, and adding edges with a weight of $+\infty$). Furthermore, this transformation can be done in $O(\log n)$ time and $O(n)$ work on the EREW PRAM. An example of such a planar layered st-graph for the planar layered digraph in Figure 3(a) is given in Figure 3(b) (with the dashed edges being the newly added edges with a weight of $+\infty$).

Hence, from now on, we assume that the given planar layered digraph is an n-vertex planar layered st-graph with a k-line embedding.

We need to review some useful structures of planar st-graphs, based on the development of Tamassia and Preparata [31]. Let F be the set of faces of a planar st-graph $G = (V, E)$.

Definition 2 *Four functions $left(\cdot)$, $right(\cdot)$, $low(\cdot)$, and $high(\cdot)$ are defined on the set $V \cup E \cup F$, as follows.*

1. *For a vertex $v \in V$, $left(v)$ (resp., $right(v)$) is the face in F that is to the left (resp., right) of v and separates the incoming edges of v from its outgoing edges; $low(v) = high(v) = v$.*
2. *For an edge $e \in E$ from vertex u to vertex v, $left(e)$ (resp., $right(e)$) is the face in F that is immediately to the left (resp., right) of e; $low(e) = u$, and $high(e) = v$.*
3. *For a face $f \in F$, $left(f) = right(f) = f$; $low(f)$ (resp., $high(f)$) is the common starting (resp., ending) vertex of the two bounding paths of f.*

The external face f' of G is conceptually partitioned into two faces: The "left" and "right" external faces of G, with the left (resp., right) external face being to the left (resp., right) of the left (resp., right) bounding path of f'.

Definition 3 *The dual graph $G^* = (V^*, E^*)$ of a planar st-graph $G = (V, E)$ is the directed graph obtained as follows: $V^* = F \cup \{s^*, t^*\}$, where s^* (resp., t^*) is the left (resp., right) external face of G. For every edge $e \in E$, there is an edge $(left(e), right(e)) \in E^*$ from $left(e)$ to $right(e)$.*

It is easy to see that G^* is also a planar st-graph with source s^* and sink t^*.

Definition 4 *Two partial orders, denoted by $\uparrow$ and $\rightarrow$, are defined on $V \cup E \cup F$ of G as follows: For any $x, y \in V \cup E \cup F$, it is said that x is* below *y (denoted by $x \uparrow y$) if there is a path from $high(x)$ to $low(y)$ in G, and that x is* to the left of *y (denoted by $x \rightarrow y$) if there is a path from $right(x)$ to $left(y)$ in the dual graph G^* of G.*

Exactly one of the following holds for any $x, y \in V \cup E \cup F$: $x \uparrow y$, $y \uparrow x$, $x \rightarrow y$, or $y \rightarrow x$ [31].

Definition 5 *Two total orders, denoted by $<_L$ and $<_R$, are defined on $V \cup E \cup F$ of G as follows: For any $x, y \in V \cup E \cup F$, $x <_L y$ iff $x \uparrow y$ or $x \rightarrow y$; $x <_R y$ iff $x \uparrow y$ or $y \rightarrow x$. The sequence of all elements in $V \cup E \cup F$ sorted according to $<_L$ (resp., $<_R$) is called the* left sequence *(resp.,* right sequence*) of G.*

The two total orders $<_L$ and $<_R$ are very useful. For example, one can make use of the fact that there is a path from a vertex v to another vertex w in a planar st-graph G iff v precedes w in *both* the left and right sequences of G [31]. Tamassia and Vitter [32] gave an $O(\log n)$ time, $O(n)$ work EREW PRAM algorithm for computing the two total orders $<_L$ and $<_R$.

WLOG, we assume that the source vertex for our single-source shortest path and depth-first search computation is the source s of the input planar st-graph G. When the source vertex for these

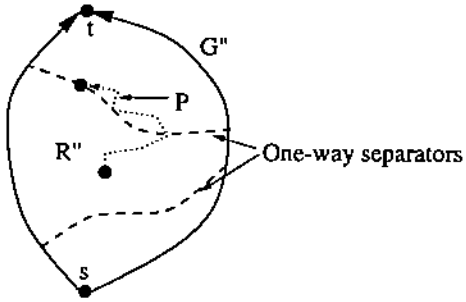

Figure 4: A shortest path P between two vertices of R'' may go outside R''.

problems is $v \neq s$, one can reduce such a problem to one on a (possibly smaller) planar st-graph whose source is v, in $O(\log n)$ time and $O(n)$ work on the EREW PRAM (by using the parallel transitive closure result of [32]).

A number of results on computing various separators for planar graphs are known (e.g., see [9, 19, 22, 23, 24]). The graph partitioning schemes of our parallel shortest path algorithms are based on two types of separators specifically for planar st-graphs. These planar st-graph separators have some special properties. One type of the separators, called *one-way separators*, was introduced by Sairam, Tamassia, and Vitter [28]. The following definition reviews and generalizes the idea of one-way separators.

Definition 6 *Let S be a subset of vertices of a directed graph $G' = (V', E')$ such that removing S and the edges adjacent to any vertex of S from G' partitions $V' - S$ into subsets V_1, V_2, ..., V_h, with $h > 1$. For any subset W of V', let $G'(W)$ be the subgraph of G' whose vertices are the vertices of W and whose edges are the edges of G' connecting vertices of W. S is called a* one-way separator *of G' if for any directed path P in G', one of the following holds for every V_i: 1) $P \cap G'(V_i \cup S)$ is empty, 2) $P \cap G'(V_i \cup S)$ is a connected subpath of P, 3) $P \cap G'(V_i)$ is a connected subpath of P. S is called a* strict one-way separator *of G' if S is a one-way separator of G' and if for any directed path P in G', $P \cap G'(S)$ is either empty or a connected subpath of P.*

Intuitively, for a one-way separator S of the graph G', a directed path in G' can "cross" the subgraph $G'(S)$ at most once, but may enter and leave $G'(S)$ multiple times (without crossing it). In comparison, for a strict one-way separator S, a directed path in G' can enter and leave $G'(S)$ at most once. Computationally, these two types of separators have quite different consequences. For a region R' of a graph G' that is bounded by strict one-way separators, all directed paths (and hence shortest paths) between any two vertices of R' never go outside R' (although these paths can touch the boundary vertices of R'). Therefore, the shortest paths in G' between any two vertices of R' can be computed simply from the region R' of G'. However, for a region R'' of a graph G'' that is bounded by one-way separators, a directed path P (and hence a shortest path) that is between two vertices of R'' and that has an end vertex on the boundary of R'' need not stay insides the boundary of R'' — such a path may pass through vertices of G'' outside R'' (see Figure 4 for an example). Therefore, the shortest paths between two vertices of R'' and with an end vertex on the

boundary of R'' that are computed only from the region R'' may not be true shortest paths in G''. This makes the shortest path computation using one-way separators somewhat more difficult.

As in [1, 2, 28], our algorithms involve multiplying special kinds of matrices (matrices with monotonicity, or Monge, property), in the $(\max, +)$ closed semi-ring, i.e., $(M' * M'')(i,j) = \max_k \{M'(i,k) + M''(k,j)\}$. Although the situation depicted in Figure 2 (b) implies that the structure that gives rise to such matrices is not always available, the fact that we can deal with the situation in Figure 2 (a) will be useful.

For two disjoint vertex sets A and B on the boundary of a region R of G, let matrix M_{AB} contain the lengths of the shortest paths that start in A and end in B (by convention, these paths go through the vertices of G on their way from A to B).

Definition 7 *Let X and Y be two disjoint vertex sets on the boundary of a region R of G, each totally ordered in some way (i.e., using one of the orders in Definition 5) , and such that the rows (resp., columns) of the matrix M_{XY} are as in the ordering for X (resp., Y). Then, matrix M_{XY} is* Monge *iff for any two successive vertices p, p' in X and two successive vertices q, q' in Y, we have $M_{XY}(p,q) + M_{XY}(p',q') \leq M_{XY}(p,q') + M_{XY}(p',q)$.*

The following well-known lemma [2, 1] will also be useful.

Lemma 1 *Assume that matrices M_{XY} and M_{YZ} are Monge, with $|X| = c_1|Y| \leq c_2|Z|$ for some positive constants c_1 and c_2. Then $M_{XY} * M_{YZ}$ can be computed in $O(\log |Y|)$ time and $O(|X||Z|)$ work in the CREW PRAM model.*

3 All-Pairs Shortest Paths in Planar *st*-Graphs

This section presents our $O(\log^2 n)$ time, $O(n^2)$ work CREW PRAM algorithm for computing all-pairs shortest paths in planar *st*-graphs. Our parallel algorithm uses a divide-and-conquer approach. A key to this divide-and-conquer algorithm is a partitioning scheme that makes use of strict one-way separators. We will discuss first our graph partitioning scheme, and then our divide-and-conquer algorithm.

3.1 Strict One-Way Separators

The strict one-way separators we use are based on a geometric structure that is described by the following visibility representation for planar *st*-graphs.

Definition 8 *A* visibility representation *$VR(G)$ for a planar st-graph G is a mapping of G in the plane as follows: Every vertex u of G is mapped to a horizontal line segment $VR_h(u)$, and every directed edge (u,w) of G is mapped to a vertical line segment $VR_v((u,w))$ whose lower (resp., upper) endpoint is on the horizontal segment $VR_h(u)$ (resp., $VR_h(w)$) such that $VR_v((u,w))$ does not intersect any other horizontal segment of $VR(G)$ (see Figure 5 for an example).*

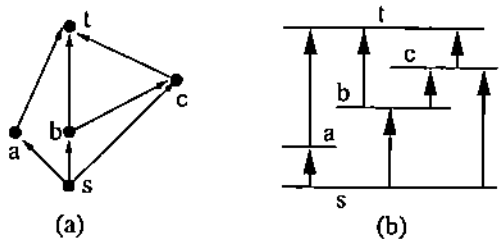

Figure 5: (a) A planar st-graph. (b) A visibility representation for the graph of (a).

It has been shown in [32] that a visibility representation $VR(G)$ of a planar st-graph G can be computed in $O(\log n)$ time and $O(n)$ work on the EREW PRAM, such that all the y-coordinates of the horizontal segments of $VR(G)$ are distinct integers. Actually, the sequence of the horizontal segments of $VR(G)$ in the increasing order of their y-coordinates corresponds to a topologically sorted sequence of the vertices of G [32]. WLOG, we assume that such a visibility representation $VR(G)$ in the plane is already available for the planar st-graph G we use.

A strict one-way separator S_G for an n-vertex planar st-graph G is defined as follows: Let $L(S_G)$ be a horizontal line in the plane such that $n/2$ horizontal segments of $VR(G)$ are strictly above (resp., strictly below) $L(S_G)$. Then $L(S_G)$ intersects the interior of a number of vertical segments of $VR(G)$. If $L(S_G)$ intersects the vertical segment $VR_v(e)$ for an edge e of G, then we assume that $L(S_G)$ intersects $VR_v(e)$ at an interior point which corresponds to an *artificial-vertex* $av(e)$ of G. These artificial-vertices are treated like ordinary vertices of G in the following way: If $av(e)$ is an artificial-vertex on the original edge $e = (u, v)$ of G, then it is as if the edge (u, v) is replaced by edges $(u, av(e))$ and $(av(e), v)$ in G. However, artificial-vertices have no representation in $VR(G)$, and hence they do not affect our graph partitioning scheme (i.e., we compute strict one-way separators based only on the original information of the graph G). (As to be shown later, artificial-vertices are useful in our shortest path computation.) Let S_G be the sequence of such artificial-vertices created by the line $L(S_G)$, such that S_G is in the increasing order of the x-coordinates of the intersection points between $L(S_G)$ and the corresponding vertical segments of $VR(G)$.

Lemma 2 *The sorted sequence S_G of all artificial-vertices created by the horizontal line $L(S_G)$ on $VR(G)$ is a strict one-way separator for the planar st-graph G, such that S_G partitions G into two regions (for which the number of original vertices differs by at most one). Furthermore, for any two consecutive artificial-vertices $av(e')$ and $av(e'')$ in S_G, their corresponding edges e' and e'' are on the boundary of the same face f of G, with e' (resp., e'') being on the left (resp., right) bounding path of f.*

Proof. The sequence S_G is a strict one-way separator because of the following: (1) $L(S_G)$ cuts $VR(G)$ into two parts, (2) any directed path from an artificial-vertex $av(e)$ in S_G can only go strictly upwards in $VR(G)$, and (3) each artificial-vertex is in the interior of an original edge of G. By its definition, S_G clearly partitions G into two regions, each with the same number of original vertices. The fact on any two consecutive artificial-vertices $av(e')$ and $av(e'')$ in S_G follows from the way in which a horizontal line intersects the structure of $VR(G)$. □

Actually, it is not hard to see that any directed path in G intersects at most one artificial-vertex of S_G (i.e., no directed path in G passes through two distinct artificial-vertices of S_G). From the visibility representation $VR(G)$ of G, it is easy to compute the strict one-way separator S_G of G, in $O(\log n)$ time and $O(n)$ work on the EREW PRAM. With such strict one-way separators, we have the following graph partitioning scheme for the planar st-graph G:

1. Use the strict one-way separator S_G to partition G into two regions G_1 and G_2 (with G_1 "below" G_2). Note that G_1 and G_2 each contain $n/2$ original vertices of G, but S_G and the edges of G that contain an artificial-vertex of S_G are excluded from G_1 and G_2.

2. Recursively partition each of G_1 and G_2, until every region contains exactly one vertex of G.

One can see that every region R of G generated by the above partitioning scheme is bounded by two strict one-way separators, one on a horizontal line bounding R from above and the other on another horizontal line bounding R from below. Further, every directed path (and hence shortest path) between two vertices of R stays insides R (since R is bounded by strict one-way separators).

We use a *partition tree* T_G to capture the above partitioning process on the graph G. The root of T_G is associated with G and with the strict one-way separator S_G of G. The left (resp., right) child of the root of T_G is the root of the partition tree T_{G_1} (resp., T_{G_2}) for the lower (resp., upper) region G_1 (resp., G_2) of G. Note that the leaves of T_G, in the left-to-right order, correspond to the horizontal segments of $VR(G)$ in the increasing order of their y-coordinates. Clearly, the number of levels of the tree T_G is $O(\log n)$. Since computing the strict one-way separator S_G of G takes $O(\log n)$ time and $O(n)$ work, T_G can be obtained in altogether $O(\log^2 n)$ time and $O(n \log n)$ work on the EREW PRAM.

Lemma 3 *Let z be a node of the tree T_G that is associated with a subgraph R of G and with the strict one-way separator S_R of R. Let R contain r original vertices of G. Then the size of S_R can be as big as $O(r)$ and as small as zero.*

Proof. Since R is an r-vertex planar graph, $|S_R|$ can certainly be $O(r)$ (i.e., the horizontal line $L(S_R)$ cuts $O(r)$ edges of R in the visibility representation $VR(G)$ of G). On the other hand, it is also possible that $|S_R| = 0$ since there may be no edge connecting the two regions into which S_R partitions R. □

We maintain T_G *explicitly* for our shortest path computation. Every node of T_G stores certain information: The root $root(T_G)$ of T_G stores G and S_G, the left child of $root(T_G)$ stores G_1 and S_{G_1}, etc. Some shortest path information is also stored at the nodes of T_G (more on this later).

3.2 All-Pairs Shortest Path Computation

We are now ready to discuss the parallel computation for all-pairs shortest paths in the planar st-graph G. Note that the partitioning scheme in Subsection 3.1 puts exactly one artificial-vertex $av(e)$ on every original edge e of G. Hence G has $O(n)$ artificial-vertices. We shall be computing the $O(n^2)$ all-pairs shortest paths between the $O(n)$ original and artificial vertices of G.

To simplify the computation, we need to reduce the planar st-graph G (with weighted edges) to another planar st-graph with weighted vertices: (1) For every original vertex v of G, let the weight of v be zero, (2) for every artificial-vertex $av(e)$ of G, let the weight of $av(e)$ be the weight of the corresponding original edge e of G, and (3) let the weights of all edges of G be zero. This reduction is easy to perform. We still let G denote the planar st-graph so resulted.

The two lemmas below form the basis for achieving efficiency in our shortest path computation.

Lemma 4 *Let S_R be the strict one-way separator for a region R of G. For any vertex v of G, if there are two directed paths in G, one between v and an artificial-vertex $av(e)$ of S_R and the other between v and another artificial-vertex $av(e')$ of S_R (with $av(e)$ preceding $av(e')$ in S_R), then there is a directed path in G between v and every artificial-vertex of S_R that is between $av(e)$ and $av(e')$.*

Proof. WLOG, assume that the two directed paths are from the vertex v to the artificial-vertices of S_R (if any). Let P_1 (resp., P_2) be a directed path from v to the sink t of G passing through $av(e)$ (resp., $av(e')$). Then every artificial-vertex $av(e'')$ of S_R that is between $av(e)$ and $av(e')$ is insides the region of G that is bounded by P_1 and P_2. Since there is a directed path P_3 from the source s of G to $av(e'')$ in G, P_3 must intersect a vertex on either P_1 or P_2. This implies that there is a directed path in G from v to $av(e'')$. □

Lemma 4 helps us decide for which pairs of artificial-vertices of G (on various strict one-way separators generated by our graph partitioning scheme in Subsection 3.1) we need to compute shortest paths. Note that a query on whether there is a directed path from a vertex v to another vertex w in G can be answered in $O(1)$ work after an $O(\log n)$ time, $O(n)$ work EREW PRAM preprocessing [32].

Let $SP_R(u, v)$ denote a shortest path insides a graph region R from a vertex u to another vertex v. Let $Length(P)$ denote the length of a path P.

Lemma 5 *Let S' and S'' be two strict one-way separators stored in the nodes of the partition tree T_G, such that the horizontal line $L(S')$ containing S' is bellow the horizontal line $L(S'')$ containing S''. Let a and b be two artificial-vertices of S' with a preceding b in S', and let c and d be two artificial-vertices of S'' with c preceding d in S''. Then the shortest path $SP_G(a, d)$ intersects the shortest path $SP_G(b, c)$ at a vertex of G (see Figure 6). Further, $Length(SP_G(a, c)) + Length(SP_G(b, d)) \leq Length(SP_G(a, d)) + Length(SP_G(b, c))$.*

Proof. To prove that the shortest paths $SP_G(a, d)$ and $SP_G(b, c)$ intersect each other at a vertex of G, let $P(s, a)$ (resp., $P(d, t)$) be a directed path in G from the source s of G to a (resp., from d to the sink t of G). Then the vertices b and c are on opposite sides of the upward directed path $Q = P(s, a) \cup SP_G(a, d) \cup P(d, t)$. Hence the shortest path $SP_G(b, c)$ must cross the path Q. Since $SP_G(b, c)$ goes upwards in the visibility representation $VR(G)$ of G, such a crossing cannot be below the line $L(S'')$ and cannot be above the line $L(S')$. Therefore $SP_G(a, d)$ and $SP_G(b, c)$ intersect each other (say) at a vertex h of G. To show that $Length(SP_G(a, c)) + Length(SP_G(b, d)) \leq Length(SP_G(a, d)) + Length(SP_G(b, c))$, we only need to point out that $Length(SP_G(a, c))$

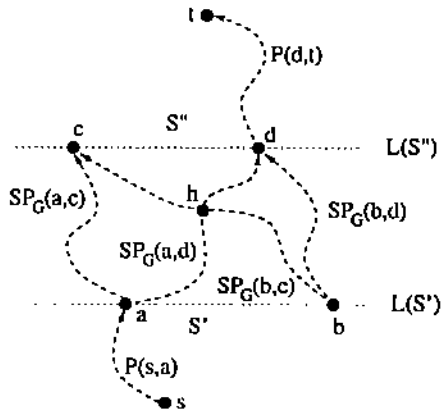

Figure 6: Illustrating Lemma 5.

$\leq$ $Length(SP_G(a,h))$ + $Length(SP_G(h,c))$ and that $Length(SP_G(b,d)) \leq Length(SP_G(b,h))$ + $Length(SP_G(h,d))$ (see Figure 6). □

Lemma 5 ensures that the length matrices for shortest paths between artificial-vertices on any two strict one-way separators generated by our graph partitioning scheme in Subsection 3.1 have the monotonicity property [1, 2]. Therefore, we can use the efficient parallel algorithms for monotone matrix multiplications in [1, 2] to compute shortest paths in planar *st*-graphs.

The main steps of our all-pairs shortest path algorithm are as follows:

1. A bottom-up stage on the partition tree T_G. In this stage, for every region R in T_G, we compute and store the shortest path lengths from the artificial-vertices on the lower boundary of R to the artificial-vertices on the upper boundary of R.

2. A top-down stage on the partition tree T_G. In this stage, for each level k in T_G and for each two regions R' and R'' on level k, such that R' is "bellow" R'', we compute the shortest path lengths from the artificial-vertices on the upper boundary of R' to the artificial-vertices on the lower boundary of R''. Thus, at the end of this stage, we have computed the all-pairs shortest path lengths between the artificial vertices in G.

3. An update stage, to obtain the shortest path lengths between original vertices of G.

The computation of the first stage proceeds upwards level by level on T_G, starting from the leaves of T_G. Suppose that the computation reaches a node z of T_G at the current level. Let z be associated with a graph region R of G and with the strict one-way separator S_R. Then S_R is the "common boundary" of the two regions associated with the two children of z in T_G. The computation at the left child of z has obtained and stored shortest path lengths from the artificial-vertices on the lower boundary of R to the artificial-vertices on S_R, and the computation at the right child of z has obtained and stored shortest path lengths from the artificial-vertices on S_R to the artificial-vertices on the upper boundary of R. The computation at z then obtains and stores the shortest path lengths from the artificial-vertices on the lower boundary of R to the artificial-vertices on the upper boundary of R, using the shortest path information stored at the two children

of z. This shortest path computation is based on Lemmas 4 and 5.

After the first stage reaches the root of T_G, the second stage takes over. The computation of the second stage proceeds downward level by level on T_G. Let u and v be any two nodes of T_G at the current level, such that u (resp., v) is associated with a region R' (resp., R'') of G and with the strict one-way separator $S_{R'}$ (resp., $S_{R''}$), and such that R' is "below" R''. Then the shortest path lengths from the artificial-vertices on the upper boundary of R' to the artificial-vertices on the lower boundary of R'' are computed. This shortest path computation gets help from the shortest path information already computed and stored at the parent node u' of u and at the parent node v' of v, and is also based on Lemmas 4 and 5. For a node $x \in T_G$, let $S_l(x)$ (resp., $S_r(x)$) be the one-way separator bounding the region R_x, associated with x, from bellow (resp., above). For two one-way separators S_1 and S_2, let $M[S_1, S_2]$ be the matrix of shortest path lengths from the artificial vertices on S_1 to the artificial vertices on S_2. There are four possible cases to consider for every pair of nodes u and v at the current level, depending on whether u (resp., v) is the left or right child of its parent node:

Case 1: u is the right child of u' and v is the left child of v'. Then, the shortest path lengths were already computed at the previous level.

Case 2: u is the right child of u' and v is the right child of v'. Then, the shortest path matrix $M[S_r(u), S_l(v)]$ is computed as a monotone matrix multiplication between the shortest path matrix $M[S_r(u'), S_l(v')]$, stored at u' (computed earlier in the second stage), and the shortest path matrix $M[S_l(x), S_r(x)]$, stored at the left child x of v' (computed in the first stage).

Case 3: u is the left child of u' and v is the left child of v'. Then, the shortest path matrix $M[S_r(u), S_l(v)]$ is computed as a monotone matrix multiplication between the shortest path matrix $M[S_l(x), S_r(x)]$, stored at the right child x of u' (computed in the first stage) and the shortest path matrix $M[S_r(u'), S_l(v')]$, stored at u' (computed earlier in the second stage).

Case 4: u is the left child of u' and v is the right child of v'. Then, the shortest path matrix $M[S_r(u), S_l(v)]$ is computed as two monotone matrix multiplications. The first multiplication, between the shortest path matrix $M[S_l(x), S_r(x)]$, stored at the right child x of u' (computed in the first stage) and the shortest path matrix $M[S_r(u'), S_l(v')]$, stored at u' (computed earlier in the second stage), computes the shortest path matrix $M[S_l(x), S_l(v')]$. The second multiplication, between the shortest path matrix $M[S_l(x), S_l(v')]$ and the shortest path matrix $M[S_l(y), S_r(y)]$, stored at the left child y of v' (computed in the first stage), gives the shortest path matrix $M[S_r(u), S_l(v)]$.

To obtain the shortest path lengths between the original vertices of G, we only need to observe that the shortest path length between two artificial vertices $av(e)$ and $av(e')$, on edges e and e'

respectively, is also the shortest path length between the original vertices z and z', where z is the lower vertex of $av(e)$ and z' is the upper vertex of $av(e')$ in $VR(G)$.

Our parallel all-pairs shortest path algorithm takes altogether $O(\log^2 n)$ time and $O(n^2)$ work on the CREW PRAM. To obtain this result, we perform an amortized analysis over all the $O(\log n)$ levels in T_G, in order to bound the total work of our algorithm. Since the work in the top-down stage dominates the work in the bottom-up stage, we only need to analyze the work in the top-down stage. Recall that the total number of artificial vertices (between which we compute shortest path lengths) is $O(n)$ and the work to compute the shortest path lengths between the boundary vertices of two regions R' and R'', such that R' (resp., R'') has k' (resp., k'') boundary vertices, is $O(k'k'')$. Then, the work in this stage (over all the $O(\log n)$ levels in T_G) is $O(\sum_{i,j=1,i\neq j}^{n} k_i k_j)$, where $\sum_{i=1}^{n} k_i = O(n)$ and $\sum_{j=1}^{n} k_j = O(n)$. Since $O(\sum_{i,j=1,i\neq j}^{n} k_i k_j) = O((\sum_{i=1}^{n} k_i)(\sum_{j=1}^{n} k_j)) = O(n^2)$, it follows that the work in the top-down stage is $O(n^2)$.

Once the all-pairs path lengths are available, the actual single-source shortest path tree SPT_u, for every source vertex $u \in G$, is easy to generate in $O(\log n)$ time and $O(n)$ work. This computation is based on the fact that, for every vertex v with $Length(SP_G(u,v)) \neq +\infty$ (thus, $v \in SPT_u$), the equality $Length(SP_G(u,v)) = Length(SP_G(u,w)) + Length(w,v)$ holds for at least one vertex w, such that (w,v) is an incoming edge of the vertex v.

Note that our parallel all-pairs shortest path algorithm on planar st-graphs, when applied to computing all-pairs shortest paths in the planar directed acyclic grid graphs considered in [1, 2], yields an $O(\log^2 n)$ time, $O(n^2)$ work CREW PRAM solution.

4 Single-Source Shortest Paths in Planar Layered st-Graphs

This section presents our $O(\log^2 n)$ time, $O(n\log n)$ work CREW PRAM algorithm for computing single-source shortest paths in planar layered st-graphs. As the all-pairs algorithm in Section 3, our parallel single-source algorithm uses a divide-and-conquer approach. A key to this divide-and-conquer algorithm is a partitioning scheme that makes use of one-way separators (rather than the strict one-way separators in Section 3).

One might wonder why we use one-way separators instead of strict one-way separators to compute in parallel single-source shortest paths in planar layered st-graphs. Recall from Lemma 3 that the size of a strict one-way separator S_R for a subgraph R of a planar st-graph can be proportional to the size of R. This leads to a parallel shortest path algorithm whose work bound is at least quadratic. Such a parallel algorithm would certainly be too expensive for the single-source problem on planar layered st-graphs, whose sequential time bound is only linear.

The one-way separators we use in this section are a modified version of those used in [28]. The size of such a one-way separator for a graph region is proportional to the *square-root* of the size of the region. Hence, these one-way separators are more likely to yield an efficient parallel single-source algorithm. On the other hand, as discussed in Section 2, one-way separators do not ensure as much locality as strict one-way separators for shortest path computation (i.e., true shortest paths

may go outside a region bounded by one-way separators). Therefore, shortest path computation based on one-way separators must be done with much more care in order to achieve the desired efficiency.

Lemma 6 ([28]) *Every n-vertex planar layered st-graph G has a one-way separator X such that*

(1) *$|X| \leq c\sqrt{n}$ for some positive constant c, and*

(2) *X partitions $G - X$ into at most four regions, such that the number of vertices of each region is no bigger than $2n/3$.*

Furthermore, such a separator can be computed in $O(\log n)$ time and $O(n)$ work on the EREW PRAM if a k-line embedding of G is given.

Proof. See Theorems 2 and 3 in [28]. □

We modify the algorithm for computing a one-way separator in [28] to generate a slightly different one-way separator as shown in the next lemma.

Lemma 7 *Given a k-line embedding of an n-vertex planar layered st-graph G, a one-way separator X' of G can be obtained in $O(\log n)$ time and $O(n)$ work on the EREW PRAM, such that*

(1') *$|X'| \leq c\sqrt{n}$ for some positive constant c, and*

(2') *X' partitions $G - X'$ into exactly* two *regions A' and B', such that neither A' nor B' has more than $2n/3$ vertices.*

Proof. We first review briefly the properties of the one-way separator used in [28] (the reader is referred to [28] for more details). The one-way separator X computed in Theorem 3 of [28] consists of two layers L_i and L_l of the graph G, and possibly a special directed path p in the portion M of G between L_i and L_l (see Figure 7). The two layers L_i and L_l together partition G into at most three regions A, B, and M. If M has less than $2n/3$ vertices, then no path p is needed, and $X = L_i \cup L_l$. If M has at least $2n/3$ vertices, then a path p in M is computed as follows: Transform M into a planar layered st-graph (by mainly adding a source and a sink to M); let m be the "middle" vertex of the vertices of M in the left sequence of M; obtain a path p by going up from m and following the leftmost outgoing edges until L_l is reached, and by going down from m and following the rightmost incoming edges until L_i is reached. The path p partitions M into two regions C and D, each with the same number of vertices (see Figure 7). Let $X = L_i \cup L_l \cup p$. X so obtained is a one-way separator with the following properties:

- None of A and B has more than $2n/3$ vertices, and none of C and D has more than $n/2$ vertices.
- $|X| = |L_i \cup L_l \cup p| \leq c\sqrt{n}$ for some positive constant c.

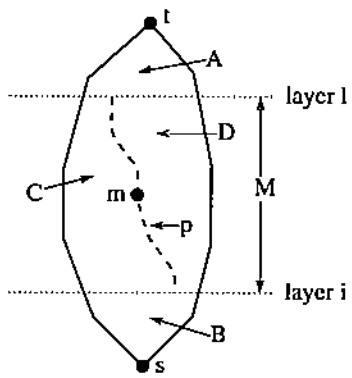

Figure 7: Illustrating the one-way separator.

We modify the algorithm in [28] so that it computes a one-way separator X' that partitions G into two regions A' and B'. Our algorithm consists of the following steps: (a) Use layers L_i and L_l to partition G into A, B, and M as in Figure 7. (b) If either A or B has at least $n/3$ vertices (say, A has $\geq n/3$ vertices), then let $A' = A$ and $B' = M \cup B$. (c) Otherwise (none of A and B has $\geq n/3$ vertices), use the path p to partition M into C and D, and let $A' = A \cup D$ and $B' = B \cup C$.

The parallel complexity bounds of our algorithm are just the same as those in Lemma 6. We now claim that (i) the "common boundary" X' between A' and B' is a one-way separator with $|X'| \leq c\sqrt{n}$, and (ii) none of A' and B' has more than $2n/3$ vertices.

> **Proof of the Claims:** It is clear that X' separates $G - X'$ into two regions A' and B'. Claim (i) follows from the proof of Theorem 2 of [28], because $X' \subseteq L_i \cup L_l \cup p$. We now prove Claim (ii). Note that A' and B' are obtained from Steps (b) or (c). In Step (b), suppose A has $\geq n/3$ vertices; then Claim (ii) easily follows because $A = A'$ and A has $\leq 2n/3$ vertices. In Step (c) (none of A and B has $\geq n/3$ vertices), let $2n_{CD}$ be the number of vertices in M and let n_A (resp., n_B) be the number of vertices in A (resp., B). Then, $(n_A + n_{CD}) + (n_B + n_{CD}) \leq n$. Assume that A' has more than $2n/3$ vertices (the proof for B' is similar), that is, $n_A + n_{CD} > 2n/3$. Then, since none of A and B has $\geq n/3$ vertices, $(n_A + n_{CD}) + (n_B + n_{CD}) > n$, which is a contradiction. Thus, Claim (ii) follows.

This completes the proof of this lemma. □

Note that although the algorithm in [28] finds a one-way separator that partitions the graph G into at most four regions (each of which has no more than $2n/3$ vertices), every such region may have as few as $O(1)$ vertices. In contrast, we compute a one-way separator that partitions the graph G into exactly two regions, whose sizes are nicely bounded from above and from below. This simplifies the subsequent shortest path computation.

Based on our one-way separators, we obtain a graph partitioning scheme. As in Section 3, we maintain explicitly a partition tree T_G that captures the partitioning process on the graph G.

The main steps of our single-source shortest path algorithm are as follows:

1. A bottom-up stage on the partition tree T_G. In this stage, when the level-by-level computation reaches a node of T_G that is associated with a graph region R of G, we do the following:

   (a) Compute the all-pairs shortest path lengths between all boundary vertices of the region R that do not go outside R.

   (b) Compute the all-pairs shortest path lengths between all boundary vertices of the region R which may go through a neighboring region of R.

   The computation in this stage is based on the parallel algorithms in [1, 2] for multiplying monotone matrices of sizes $O(m) \times O(m)$.

2. A top-down stage on the partition tree T_G. In this stage, when the level-by-level computation reaches a node of T_G that is associated with a graph region R of G, the shortest paths from the source s of G to all boundary vertices of R are computed. This computation is based on the parallel algorithm of Atallah and Kosaraju [3] for searching monotone matrices of sizes $O(m) \times O(m)$.

Note that the shortest path algorithm in [28] also uses a bottom-up procedure (but not a top-down one) for computing a source-to-sink shortest path in G. Although the bottom-up procedure in [28] is based on one-way separators (which may allow true shortest paths to go outside a region bounded by such separators), it is able to compute correctly the source-to-sink shortest path in G. The reason for this is that such a bottom-up procedure can correctly compute shortest paths between boundary vertices of a graph region provided that the shortest paths *do not go outside that region*. Observe that the source-to-sink shortest path in G is eventually computed between two boundary vertices of a graph region that is G itself.

Our single-source algorithm also hinges on computing shortest paths between boundary vertices of the graph regions that are generated by our graph partitioning scheme based on one-way separators. However, in the single-source case, we must compute the "true" shortest paths from s to *all* vertices of G. Consider a region R' generated by our partitioning scheme such that R' is a proper subregion of G. In order to compute the "true" shortest paths from s to all the boundary vertices of R', we need to compute correctly and efficiently the "true" shortest paths between the boundary vertices of R'. But, some of these shortest paths may go outside R' (even though these paths are still inside the largest region G). See Figure 8 for an example. Our single-source algorithm must handle this difficulty carefully.

Our idea for resolving this difficulty is to divide the "candidate" shortest paths into several groups, such that the computation on each group of paths can be done by using the monotone matrix searching or multiplication algorithms in [1, 2, 3]. The correct shortest paths are then selected among their "candidate" paths.

Let $M[R, R]$ denote the matrix of shortest path lengths between the boundary vertices of a region R in T_G and let $M[s, R]$ denote the matrix of shortest path lengths from s to the boundary vertices of R.

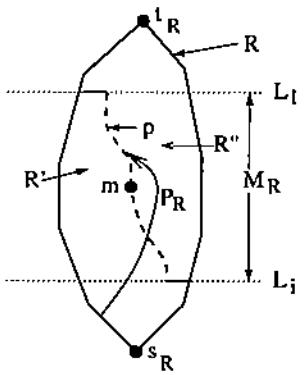

Figure 8: A shortest path P_R goes outside R' through R''.

We compute the shortest path lengths between boundary vertices in the bottom-up stage. Suppose that the bottom-up computation reaches a node u of T_G that is associated with a graph region R. Let X_R be the one way separator associated with R, and let L_i and L_l be the two horizontal layers of X_R (see Figure 8). The difficulty with our one-way separator arises when $X_R = L_i \cup L_l \cup p$, that is, when the subgraph region M_R of R between L_i and L_l has more than $2r/3$ vertices, where r is the number of vertices in R. To see why this is the case, let v and w be the left and right children of u in T_G, associated with regions R' and R'' respectively. Then, the shortest paths between the vertices on the boundary of R' (resp., R'') which stay inside R' (resp., R'') are already computed at v (resp., w). However, some shortest paths from vertices on the boundary of R' to vertices on the directed path p of X_R may go through R'' (e.g., the path P_R in Figure 8). To compute these shortest paths, we do the following:

1. From the shortest path matrix stored at R'', select the submatrix of shortest paths between pairs of vertices on p.

2. Use the shortest paths between vertices on p to compute the correct shortest paths (which may go through R'') between the vertices on the boundary of R' and the vertices on p, and update the shortest path information at v. This computation can be done as a monotone matrix multiplication between $M[R, p]$ and $M[p, p]$.

3. Choose the "better" path lengths for $M[R', R']$ from those staying inside R' and from those going through R''.

Finally, the shortest paths between vertices on R are computed. This shortest path computation can be done as a monotone matrix multiplication between $M[R', R']$ and $M[R'', R'']$.

After the first stage reaches the root of T_G, the second stage takes over. The computation of the second stage proceeds downward level by level on T_G, at each level computing the shortest path lengths from s to the boundary vertices of each region on that level.

The computation at a node $u \in T_G$ that is associated with a graph region R uses the parallel algorithm in [3] to compute $M[s, R]$. Let $M[s, R']$ be the shortest path length matrix for the region R', associated with the parent of u in T_G. From $M[s, R']$, we only need the submatrix of shortest

path lengths from s to the boundary vertices of R' that are also boundary vertices of R. It is straightforward to extract this submatrix once $M[s, R']$ is available. We refer to this submatrix as $M'[s, R']$. Note that, if $M[R, R]$ is an $m \times m$ matrix, then $M'[s, R']$ is a $1 \times m$ matrix. Then, $M[s, R]$ can be computed in $O(\log m)$ time, with $O(m)$ *EREW PRAM* processors [3], by multiplying the matrix $M'[s, R']$ with the matrix $M[R, R]$.

We now analyze the work performed by our algorithm.

1. For the bottom-up phase. In this phase, for each region R on every level k in T_G, we compute the shortest path lengths between boundary vertices of R. At level k, there are 2^k regions and the number of boundary vertices (which are vertices on some one-way separators) for each region is $O(\sqrt{n}/2^{k/2})$. Then, the work at level k is $O(n)$. Summing over all $O(\log n)$ levels in T_G, the work of our algorithm in the bottom-up phase is $O(n \log n)$.

2. For the top-down phase. In this phase, for each region R on every level k in T_G, we compute the shortest path lengths from s to the boundary vertices of R. The work at level k is $O(2^k \sqrt{n}/2^{k/2} \log(\sqrt{n}/2^{k/2}))$. Summing over all $O(\log n)$ levels in T_G, the work of our algorithm in the top-down phase is bounded from above by $O(n \log n)$.

Thus, our parallel single-source shortest path algorithm takes altogether $O(\log^2 n)$ time and $O(n \log n)$ work on the CREW PRAM.

5 Depth-First Search in Planar *st*-Graphs

Most known parallel depth-first search algorithms on planar graphs are hinged on some kinds of separator results [10, 13, 15, 16, 17, 18, 19, 29, 30]. However, the task of computing in parallel useful separators for planar graphs is quite nontrivial [9, 19, 24]. In contrast, our optimal parallel depth-first search algorithm on planar *st*-graphs does not rely on any separator result and is actually very simple.

Let s' be the root of the depth-first search tree on G. WLOG, we assume that s' is the source s of G (other cases can be handled easily). Suppose that, for every vertex v of the planar *st*-graph $G = (V, E)$, the cyclic ordering of v's outgoing edges is given in clockwise direction, starting from its leftmost outgoing edge. Let $incoming_{lm}(v)$ be the leftmost incoming edge of each vertex v of G ($incoming_{lm}(s)$ is empty for the source s of G). Let $INCOMING_{lm}(V) = \{incoming_{lm}(v) \mid v \in V\}$.

Lemma 8 *The subgraph $T = (V, INCOMING_{lm}(V))$ of the planar st-graph $G = (V, E)$ is a depth-first search tree of G rooted at the source s of G.*

Proof. We first prove that T is a spanning tree of G rooted at the source s of G, and then that it is a depth-first search tree of G. As previously, we assume that the embedding of G is such that all edges are oriented upward.

From the construction of T, each vertex u has an unique parent $parent(u)$ (the end vertex of $incoming_{lm}(u)$ which is different from u). Then, to prove that T is a spanning tree for G rooted

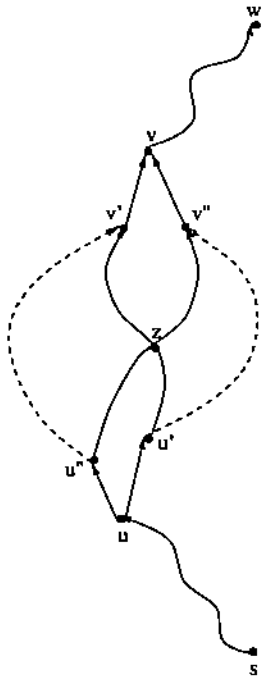

Figure 9: The path between u' and v' crosses the path between u'' and v'' at a vertex z.

at s, we only need to show that for every vertex u, there is a unique directed path from s to u in T. To prove this, for every vertex $u \in G$, we construct a path P as follows: 1) Let P be initially empty. 2) Add to P the edge $incoming_{lm}(u)$ (which is an edge in T). 3) Repeat the previous step, with u replaced by the other end vertex of $incoming_{lm}(u)$, until a vertex v is reached such that v has no incoming edges. Such a vertex v must exist and it can only be s (since only $incoming_{lm}(s)$ is empty). This completes the proof that T is a spanning tree of G rooted at s.

Let T_{DFS} be the depth-first search tree of G rooted at s, constructed by first visiting the leftmost edge of a vertex u of G, and assume that there is a vertex $w \in G$ such that the path P' from s to w in T is different from the path P'' from s to w in T_{DFS}. Then, there exist two vertices u (possibly s) and v (possibly w) such that $u,v \in P'$, $u,v \in P''$, and the path P'_{uv} from u to v in T is vertex disjoint from the path P''_{uv} from u to v in T_{DFS}. Let u' (resp., u'') be the vertex succeeding u in P' (resp., P''), and let v' (resp., v'') be the vertex preceding v in P' (resp., P'') (see Figure 9). We denote the directed edge from vertex x to vertex y as $edge(x,y)$. Then, $edge(u,u')$ is to the right of $edge(u,u'')$ ($edge(u,u'')$ is an edge on the leftmost path starting at u and passing through v) and $edge(v',v)$ is to the left of $edge(v'',v)$ ($edge(v',v)$ is the leftmost incoming edge of v). Due to the planarity of G, it results that P'_{uv} and P''_{uv} must intersect at a vertex z ($z \neq u$ and $z \neq v$) which contradicts the assumption that P'_{uv} and P''_{uv} are vertex disjoint. This completes the proof of this lemma. □

Figure 10 gives an example of the depth-first search tree T of a planar st-graph (the edges of T are dotted, and the integers are the depth-first search numbers of the vertices of the graph).

Our parallel depth-first search algorithm consists of the following steps:

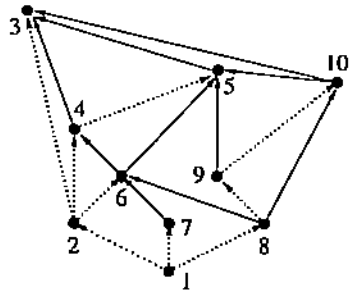

Figure 10: The depth-first search tree T of a planar st-graph.

1. For every vertex v of G, identify the edge $incoming_{lm}(v)$.
2. For every vertex v, find all its outgoing edges $e_w = (v, w)$, such that $e_w = incoming_{lm}(w)$ for some outgoing neighboring vertex w of v.
3. Form the tree $T = (V, INCOMING_{lm}(V))$ (rooted at the source s).
4. Assign preorder numbers to the vertices of T.

By Lemma 8, the preorder numbers of the vertices of the tree T are the depth-first search numbers of the vertices of G. The analysis of the above algorithm is simple. Step 1 can be easily done in $O(\log n)$ time and $O(n)$ work. Step 2 takes $O(\log n)$ time and $O(n)$ work by applying parallel list ranking technique [14] to the list of outgoing edges of every vertex v. Step 3 obtains the tree T from the outcome of Step 2. Step 4 uses the Euler tour technique [14] to compute the preorder numbers of the vertices of T. The parallel depth-first search algorithm hence takes altogether $O(\log n)$ time and $O(n)$ work on the EREW PRAM.

References

[1] A. Aggarwal and J. Park, "Notes on searching in multidimensional monotone arrays," *Proc. 29th Annual IEEE Symp. on Foundations of Computer Science*, 1988, pp. 497–512.

[2] A. Apostolico, M. J. Atallah, L. L. Larmore, and H. S. McFaddin, "Efficient parallel algorithms for string editing and related problems," *SIAM J. Comput.*, 19 (5) (1990), pp. 968–988.

[3] M. J. Atallah and S. R. Kosaraju, "An efficient parallel algorithm for the row minima of a totally monotone matrix," *J. of Algorithms*, 13 (3) (1992), pp. 394–413.

[4] R. P. Brent, "The parallel evaluation of general arithmetic expressions," *J. of the ACM*, 21 (2) (1974), pp. 201–206.

[5] E. Cohen, "Efficient parallel shortest-paths in digraphs with a separator decomposition," *J. of Algorithms*, 21 (2) (1996), pp. 331–357.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill, 1990.

[7] G. Di Battista and E. Nardelli, "An algorithm for testing planarity of hierarchical graphs," *Proc. of International Workshop on Graph-Theoretic Concepts in Computer Science*, 1986, Bernierd, pp. 277–289.

[8] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *J. of the ACM*, 34 (1987), pp. 596–615.

[9] H. Gazit and G. Miller, "A parallel algorithm for finding a separator in planar graphs," *Proc. 28th Annual IEEE Symp. on Foundations of Computer Science*, 1987, pp. 238–248.

[10] T. Hagerup, "Planar depth-first search in $O(\log n)$ parallel time," *SIAM J. Comput.*, 19 (1990), pp. 678–704.

[11] Y. Han, V. Y. Pan, and J. H. Reif, "Efficient parallel algorithms for computing all pair shortest paths in directed graphs," *Algorithmica*, 17 (4) (1997), pp. 399–415.

[12] X. He, "Efficient parallel algorithms for series parallel graphs," *J. of Algorithms*, 12 (1991), pp. 409–430.

[13] X. He and Y. Yesha, "A nearly optimal parallel algorithm for constructing depth first spanning trees in planar graphs," *SIAM J. Comput.*, 17 (1988), pp. 486–491.

[14] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, Reading, Massachusetts, 1992.

[15] J. JáJá and S. Kosaraju, "Parallel algorithms for planar graphs and related problems," *IEEE Trans. on Circuits and Systems*, 35 (1988), pp. 304–311.

[16] M. Y. Kao, "All graphs have cycle separators and planar directed depth-first search is in DNC," *Lecture Notes in Computer Science, Vol. 319, VLSI Algorithms and Architectures, Proc. of 3rd Aegean Workshop on Computing*, Springer-Verlag, New York, 1988, pp. 53–63.

[17] M. Y. Kao, "Planar strong connectivity helps in parallel depth-first search," *SIAM J. Comput.*, 24 (1995), pp. 46–62.

[18] M. Y. Kao and P. N. Klein, "Towards overcoming the transitive-closure bottleneck: Efficient parallel algorithms for planar digraphs," *J. Computer and System Sciences*, 47 (1993), pp. 459–500.

[19] M. Y. Kao, S.-H. Teng, and K. Toyama, "An optimal parallel algorithm for planar cycle separator," *Algorithmica*, 14 (1995), pp. 398–408.

[20] P. N. Klein, "Efficient parallel algorithms for chordal graphs," *Proc. 29th Annual IEEE Symp. on Foundations of Computer Science*, 1988, pp. 150–161.

[21] P. N. Klein and S. Subramanian, "A linear-processor polylog-time algorithm for shortest paths in planar graphs," *Proc. the 34th Annual Symp. on Foundations of Computer Science*, 1993, pp. 259–270.

[22] R. J. Lipton and R. E. Tarjan, "A separator theorem for planar graphs," *SIAM J. Algebraic and Discrete Methods*, 36 (2), 1979, pp. 177–189.

[23] R. J. Lipton and R. E. Tarjan, "Applications of a planar separator theorem," *SIAM J. Comput.*, 9 (3) (1980), pp. 615–627.

[24] G. Miller, "Finding small simple cycle separators for 2-connected planar graphs," *J. Computer and System Sciences*, 32 (1986), pp. 265–279.

[25] V. Pan and J. H. Reif, "Fast and efficient solution of path algebra problems," *J. Computer and System Sciences*, 38 (1989), pp. 494–510.

[26] V. Pan and J. H. Reif, "The parallel computation of minimum cost paths in graphs by stream contraction," *Information Processing Letters*, 40 (1991), pp. 79–83.

[27] J. H. Reif, "Depth-first search is inherently sequential," *Information Processing Letters*, 20 (1985), pp. 229–234.

[28] S. Sairam, R. Tamassia, and J. S. Vitter, "An efficient parallel algorithm for shortest paths in planar layered digraphs," *Algorithmica*, 14 (1995), pp. 322–339.

[29] G. E. Shannon, "A linear-processor algorithm for depth-first search in planar graphs," *Information Processing Letters*, 29 (1988), pp. 119–123.

[30] J. R. Smith, "Parallel algorithms for depth-first search I. Planar graphs," *SIAM J. Comput.*, 15 (1986), pp. 814–830.

[31] R. Tamassia and F. P. Preparata, "Dynamic maintenance of planar digraphs, with applications," *Algorithmica*, 5 (1990), pp. 509–527.

[32] R. Tamassia and J. S. Vitter, "Parallel transitive closure and point location in planar structures," *SIAM J. Comput.*, 20 (4) (1991), pp. 708–725.

[33] J. Valdes, R. E. Tarjan, and E. L. Lawler, "The recognition of series parallel digraphs," *SIAM J. Comput.*, 11 (1982), pp. 298–313.

[34] S. Whitesides, "Forms of hierarchy: A selected bibliography," *General Systems*, 14 (1969), pp. 3–15.